Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №11
дисциплины «Программирование на Python»
Вариант 9

Выполнил:
Дудкин Константин Александрович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»
направление «Программное
обеспечение средств вычислительной
техники и автоматизированных
систем»,
очная форма обучения

_____
(подпись)

Руководитель практики:
Кандидат технических наук, доцент
кафедры инфокомуникаций, доцент
Воронкин Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Тема: Работа с функциями в языке Python

Цель: Приобретение навыков по работе с функциями при написании программ с помощью языка программирования Python версии 3.x

Порядок выполнения работы

1. Проработал пример: Для примера 1 лабораторной работы №9 оформить каждую команду в виде вызова отдельной функции

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import sys
from datetime import date


1 usage  new *
def get_worker():
    """
    Запросить данные о работнике.
    """
    name = input("Фамилия и инициалы? ")
    post = input("Должность? ")
    year = int(input("Год поступления? "))
    # Создать словарь.
    return {
        'name': name,
        'post': post,
        'year': year,
    }


2 usages  new *
def display_workers(staff):
    """
    Отобразить список работников.
    """
    # Проверить, что список работников не пуст.
    if staff:
        # Заголовок таблицы.
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            *args: '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                *args: "№",
```

Рисунок 1. Код программы примера (часть 1)

```
                "Ф.И.О.",
                "Должность",
                "Год"
            )
        )
        print(line)
        # Вывести данные о всех сотрудниках.
        for idx, worker in enumerate(staff, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                    *args: idx,
                    worker.get('name', ''),
                    worker.get('post', ''),
                    worker.get('year', 0)
                )
            )
            print(line)

    else:
        print("Список работников пуст.")


1 usage  new *
def select_workers(staff, period):
    """
    Выбрать работников с заданным стажем.
    """


    # Получить текущую дату.
    today = date.today()
    # Сформировать список работников.
    result = []
    for employee in staff:
        if today.year - employee.get('year', today.year) >= pe
            result.append(employee)

```

Рисунок 2. Код программы примера (часть 2)

```
    # Возвратить список выбранных работников.
    return result


1 usage  new *
def main():
    """
    Главная функция программы.
    """
    # Список работников.
    workers = []
    # Организовать бесконечный цикл запроса команд.
    while True:
        # Запросить команду из терминала.
        command = input(">>> ").lower()
        # Выполнить действие в соответствие с командой.
        if command == 'exit':
            break
        elif command == 'add':
            # Запросить данные о работнике.
            worker = get_worker()
            # Добавить словарь в список.
            workers.append(worker)
            # Отсортировать список в случае необходимости.
            if len(workers) > 1:
                workers.sort(key=lambda item: item.get('name',
        elif command == 'list':
            # Отобразить всех работников.
            display_workers(workers)
        elif command.startswith('select '):
            # Разбить команду на части для выделения стажа.
            parts = command.split( sep: ' ', maxsplit=1)
            # Получить требуемый стаж.
            period = int(parts[1])
            # Выбрать работников с заданным стажем.
            selected = select_workers(workers, period)
            # Отобразить выбранных работников.
            display_workers(selected)
```

Рисунок 3. Код программы примера (часть 3)

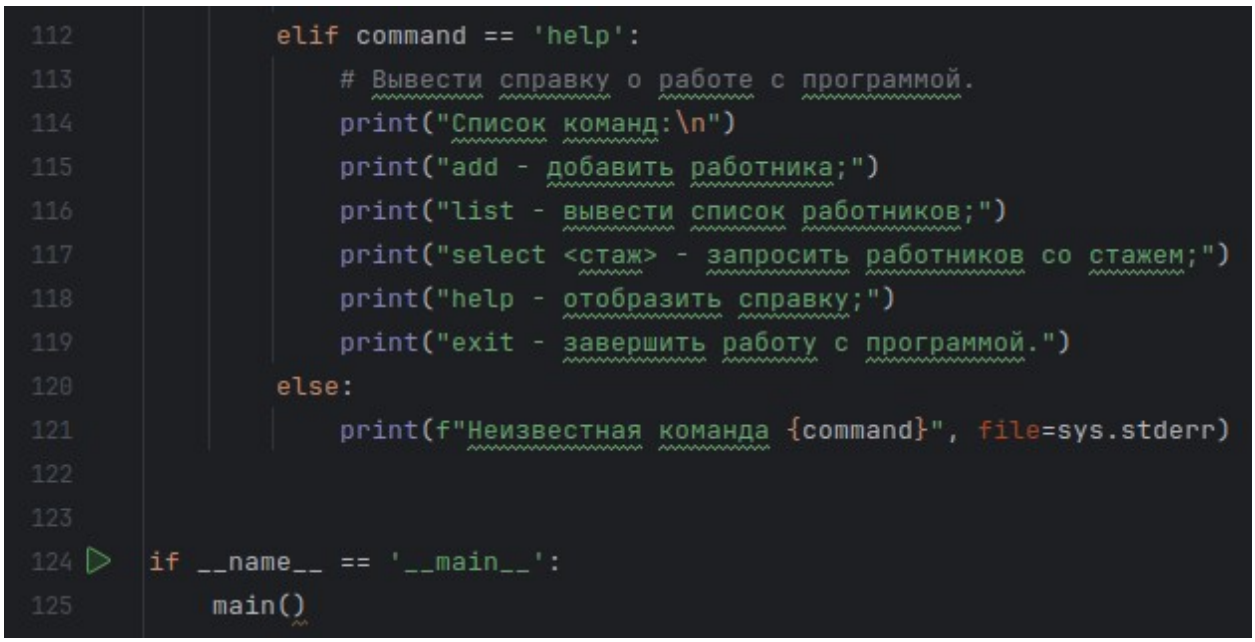


Рисунок 4. Код программы примера (часть 4)

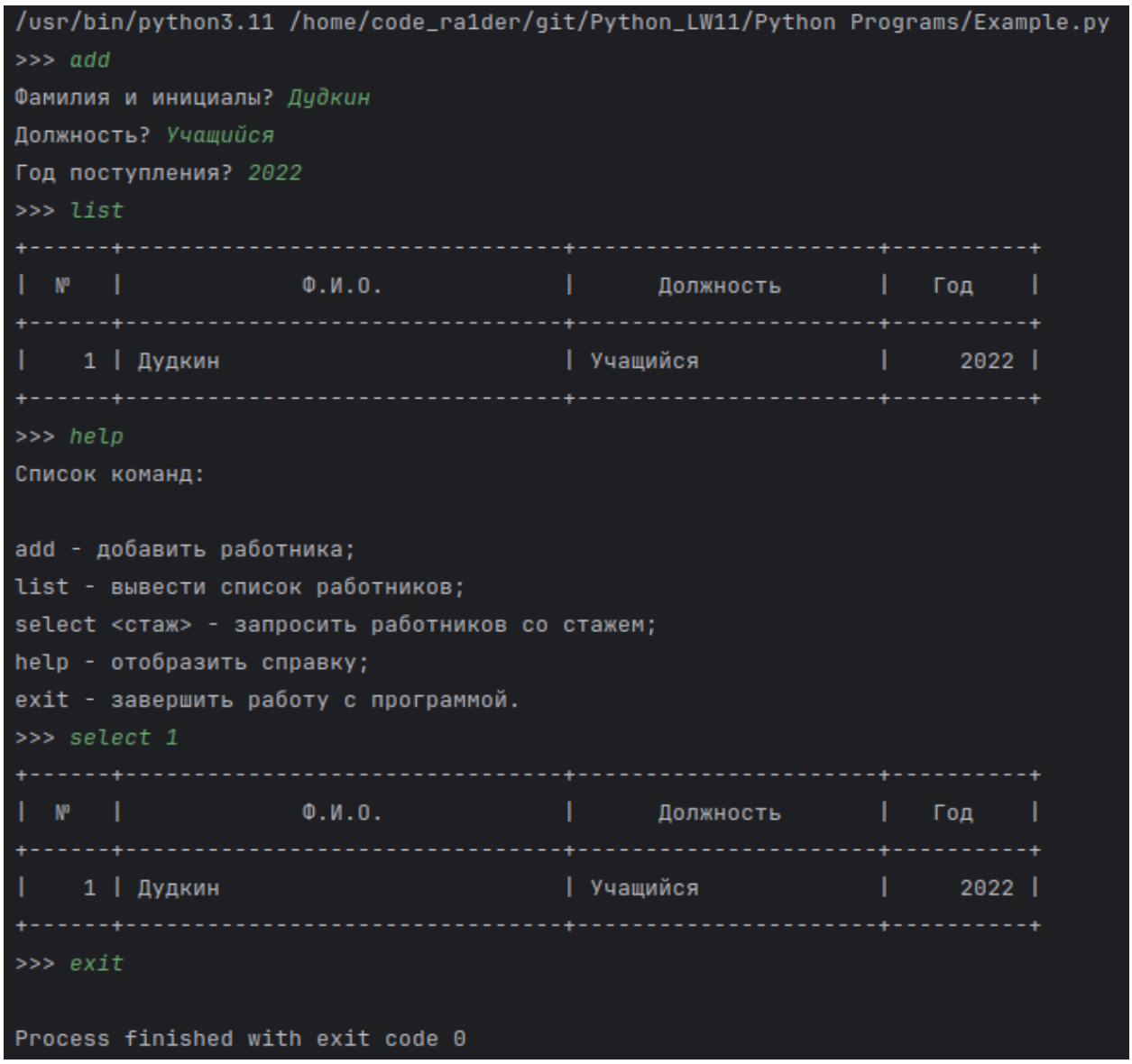


Рисунок 5. Результат работы программы

2. Выполнил основное задание №1: Основная ветка программы, не считая заголовков функций, состоит из двух строки кода. Это вызов функции test() и инструкции if __name__ == '__main__' . В ней запрашивается на ввод целое число. Если оно положительное, то вызывается функция positive(), тело которой содержит команду вывода на экран слова "Положительное". Если число отрицательное, то вызывается функция negative(), ее тело содержит выражение вывода на экран слова "Отрицательное".

Понятно, что вызов test() должен следовать после определения функций. Однако имеет ли значение порядок определения самих функций? То есть должны ли определения positive() и negative() предшествовать test()

или могут следовать после него? Проверьте вашу гипотезу, поменяв объявления функций местами. Попробуйте объяснить результат

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

1 usage  new *
def negative():
    print("Отрицательное")

1 usage  new *
def positive():
    print("Положительное")

1 usage  new *
def test():
    num = int(input("Введите целое число: "))
    if num > 0:
        positive()
    elif num < 0:
        negative()

if __name__ == '__main__':
    test()
```

Рисунок 6. Код программы задания

```
/usr/bin/python3.11 /home/code_ra1der/git/Python_LW11/Python Programs/Task1.py
Введите целое число: 7
Положительное

Process finished with exit code 0
```

```
/usr/bin/python3.11 /home/code_ra1der/git/Python_LW11/Python Programs/Task1.py
Введите целое число: -4
Отрицательное

Process finished with exit code 0
```

Рисунок 7. Результат работы программы

3. Выполнил основное задание №2: В основной ветке программы вызывается функция cylinder(), которая вычисляет площадь цилиндра. В теле cylinder() определена функция circle(), вычисляющая площадь круга по формуле $2\pi r$. В теле cylinder() у пользователя спрашивается, хочет ли он получить только площадь боковой поверхности цилиндра, которая вычисляется по формуле $2\pi rh$, или полную площадь цилиндра. В последнем случае к площади боковой поверхности цилиндра должен добавляться удвоенный результат вычислений функции circle().

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import math


1 usage  new *
def circle(radius):
    return math.pi * radius ** 2


1 usage  new *
def cylinder():
    radius = float(input("Введите радиус цилиндра: "))
    height = float(input("Введите высоту цилиндра: "))
    option = input("Хотите получить площадь боковой поверхности (s) или полную площадь (f)? ")

    if option == "s":
        lateral_area = 2 * math.pi * radius * height
        print(f"Площадь боковой поверхности цилиндра: {lateral_area}")
    elif option == "f":
        lateral_area = 2 * math.pi * radius * height
        total_area = lateral_area + 2 * circle(radius)
        print(f"Полная площадь цилиндра: {total_area}")
    else:
        print("Некорректный вариант. Выберите 's' или 'f'.")


if __name__ == '__main__':
    cylinder()
```

Рисунок 8. Код программы задания

```
/usr/bin/python3.11 /home/code_ra1der/git/Python_LW11/Python Programs/Task2.py
Введите радиус цилиндра: 5
Введите высоту цилиндра: 89
Хотите получить площадь боковой поверхности (s) или полную площадь (f)? f
Полная площадь цилиндра: 2953.0970943744055

Process finished with exit code 0
```

```
/usr/bin/python3.11 /home/code_ra1der/git/Python_LW11/Python Programs/Task2.py
Введите радиус цилиндра: 3
Введите высоту цилиндра: 7
Хотите получить площадь боковой поверхности (s) или полную площадь (f)? s
Площадь боковой поверхности цилиндра: 131.94689145077132

Process finished with exit code 0
```

Рисунок 9. Результат работы программы

4. Выполнил основное задание №3: Напишите функцию, которая считывает с клавиатуры числа и перемножает их до тех пор, пока не будет введен 0. Функция должна возвращать полученное произведение. Вызовите функцию и выведите на экран результат ее работы.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

1 usage  new *
def mult():
    result = 1
    while True:
        num = float(input("Введите число (введите 0 для завершения): "))
        if num == 0:
            break
        result *= num
    return result

if __name__ == '__main__':
    product = mult()
    print(f"Произведение введенных чисел: {product}")
```

Рисунок 10. Код программы задания

```
/usr/bin/python3.11 /home/code_ra1der/git/Python_LW11/Python Programs/Task3.py
Введите число (введите 0 для завершения): 4
Введите число (введите 0 для завершения): 7
Введите число (введите 0 для завершения): 1
Введите число (введите 0 для завершения): 34
Введите число (введите 0 для завершения): 753
Введите число (введите 0 для завершения): 0
Произведение введенных чисел: 716856.0

Process finished with exit code 0
```

Рисунок 11. Результат работы программы

5. Решите следующую задачу: Напишите программу, в которой определены следующие четыре функции:

1) Функция get_input() не имеет параметров, запрашивает ввод с клавиатуры и возвращает в основную программу полученную строку.

2) Функция test_input() имеет один параметр. В теле она проверяет, можно ли переданное ей значение преобразовать к целому числу. Если можно, возвращает логическое True. Если нельзя – False.

3) Функция str_to_int() имеет один параметр. В теле преобразовывает переданное значение к целочисленному типу. Возвращает полученное число.

4) Функция print_int() имеет один параметр. Она выводит переданное значение на экран и ничего не возвращает.

В основной ветке программы вызовите первую функцию. То, что она вернула, передайте во вторую функцию. Если вторая функция вернула True, то те же данные (из первой функции) передайте в третью функцию, а возвращенное третьей функцией значение – в четвертую

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

1 usage  new *
def get_input():
    user_input = input("Введите значение: ")
    return user_input

1 usage  new *
def test_input(value):
    try:
        int(value)
        return True
    except ValueError:
        return False

1 usage  new *
def str_to_int(value):
    return int(value)

1 usage  new *
def print_int(value):
    print(value)

if __name__ == '__main__':
    input_str = get_input()

    if test_input(input_str):
        int_value = str_to_int(input_str)
        print_int(int_value)
    else:
        print("Введенное значение не является целым числом.")
```

Рисунок 12. Код программы задачи

```
/usr/bin/python3.11 /home/code_ra1der/git/Python_LW11/Python Programs/Task4.py
Введите значение: 5
5

Process finished with exit code 0
```

Рисунок 13. Результат работы программы

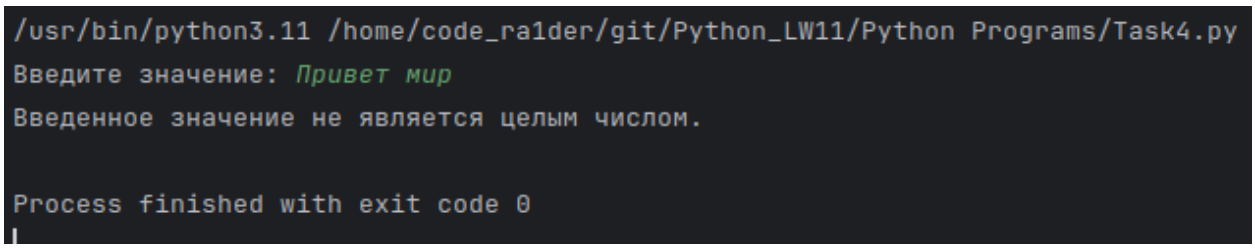
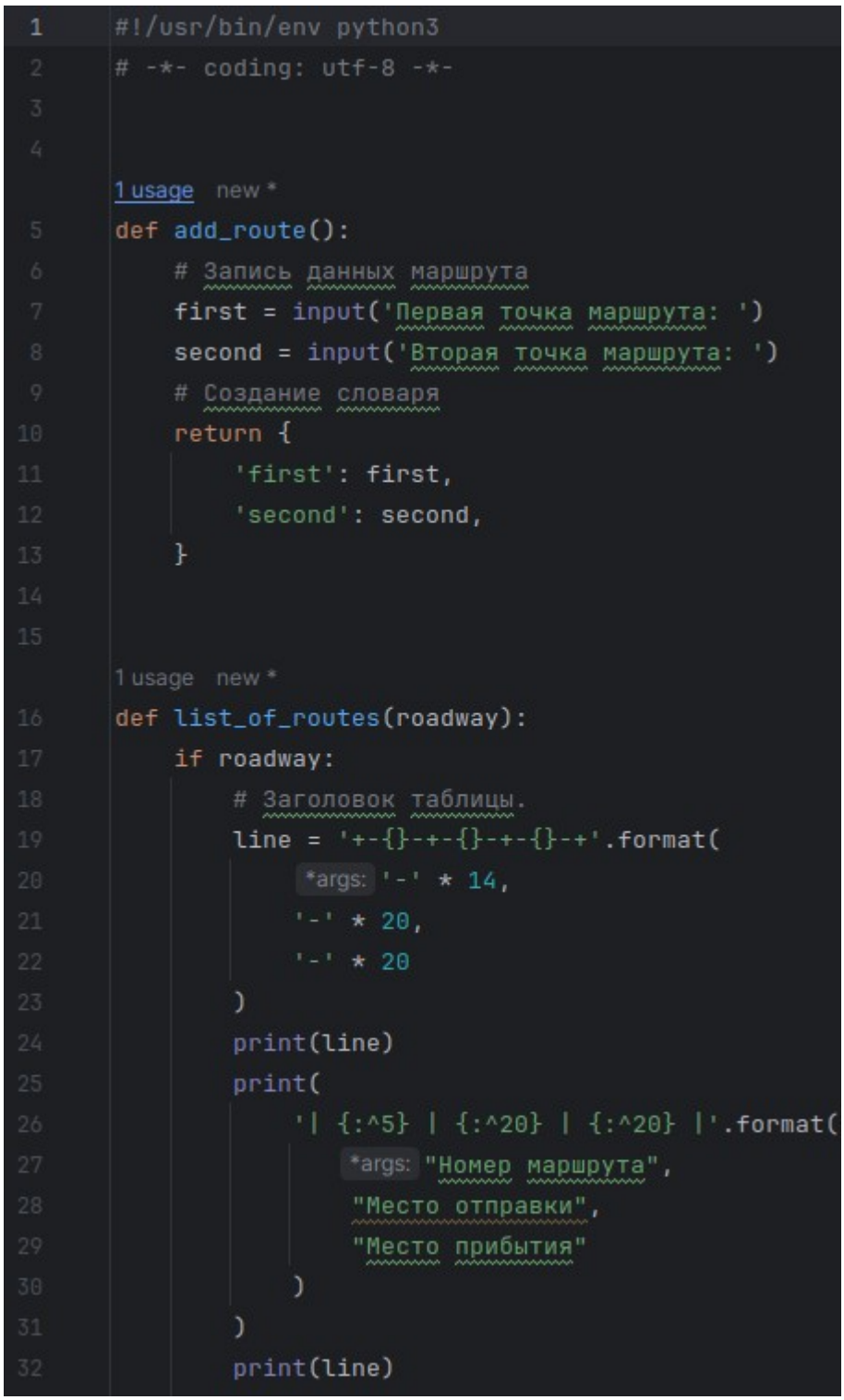


Рисунок 14. Результат работы программы («неудача»)

6. Выполнил индивидуальное задание: Решить индивидуальное задание лабораторной работы №9, оформив каждую команду в виде отдельной функции



```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


1 usage  new *
def add_route():
    # Запись данных маршрута
    first = input('Первая точка маршрута: ')
    second = input('Вторая точка маршрута: ')
    # Создание словаря
    return {
        'first': first,
        'second': second,
    }


1 usage  new *
def list_of_routes(roadway):
    if roadway:
        # Заголовок таблицы.
        line = '+-{}-+-{}-+-{}-+'.format(
            *args: '-' * 14,
            '-' * 20,
            '-' * 20
        )
        print(line)
        print(
            '| {:^5} | {:^20} | {:^20} |'.format(
                *args: "Номер маршрута",
                "Место отправки",
                "Место прибытия"
            )
        )
        print(line)
```

Рисунок 15. Код программы (часть 1)

```
33          # Вывод данных о маршрутах
34          for number, route in enumerate(roadway, 1):
35              print(
36                  '| {:<14} | {:<20} | {:<20} |'.format(
37                      *args: number,
38                      route.get('first', ''),
39                      route.get('second', '')
40                  )
41              )
42              print(line)
43      else:
44          print("Список маршрутов пуст")
45
      1 usage  new *
46    def help():
47        print('\nСписок команд:')
48        print('help - Вывести этот список')
49        print('add - Добавить маршрут')
50        print('list - Показать список маршрутов')
51        print('exit - Выйти из программы')
52
      1 usage  new *
53    def main():
54        # Список маршрутов
55        routes = []
56        # Начало бесконечного цикла команд
57        while True:
58            # Сюда вписывать команды
59            command = input('>>> ').lower()
60
61            # Команда help
62            if command == 'help':
63                help()
64
```

Рисунок 16. Код программы (часть 2)

```
        # Команда add
        elif command == 'add':
            route = add_route()
            # Добавление словаря в список
            routes.append(route)

        # Команда list
        elif command == 'list':
            list_of_routes(routes)

        # Команда exit
        elif command == 'exit':
            break

        # Другая команда/неверно введенная команда
        else:
            print(f'Неизвестная команда {command}')

if __name__ == '__main__':
    main()
```

Рисунок 17. Код программы (часть 3)

```
/usr/bin/python3.11 /home/code_ra1der/git/Python_LW11/Python Programs/Individual.py
>>> add
Первая точка маршрута: Ставрополь
Вторая точка маршрута: Ессентуки
>>> list
+----------------+----------------------+----------------------+
| Номер маршрута |    Место отправки    |    Место прибытия    |
+----------------+----------------------+----------------------+
| 1              | Ставрополь           | Ессентуки            |
+----------------+----------------------+----------------------+
>>> help

Список команд:
help - Вывести этот список
add - Добавить маршрут
list - Показать список маршрутов
exit - Выйти из программы
>>> something
Неизвестная команда something
>>> exit

Process finished with exit code 0
```

Рисунок 18. Результат работы программы

Ответы на вопросы

1. Функция в Python представляет собой обособленнй участок кода, который можно вызвать, обратившись к нему по имени, которым он был назван. Функции, в своем роде, представляют собой подпрограммы, которые встроены в саму программу и не которые не могут выполняться самостоятельно. В некоторых случаях их так и называют подпрограммами, других отличий с программами у них нет.

Функции также могут при необходимости принимать и возвращать данные. В основном это происходит с данными, полученными из самой программы, а не напрямую с клавиатуры. Все данные потом возвращаются в программу для дальнейшей из обработки в ней или другими функциями

2. def предназначен для создания функций. Сначала этот оператор закрепляет имя функции, затем после него идет основная часть, где вписываются все действия и какие данные применяются. Затем, когда все действия были расписаны, происходит возврат данных — использование оператора return, хотя бывают такие случаи, когда возврат данных либо не требуется, либо требуется только в строго определенных случаях, из-за чего этот оператор не всегда вписывается в функции

3. Локальные переменные — переменные, которые образованы в определённом блоке функции и имеющие значение только в них. Когда программа выходит из этой функции, локальные переменные стираются и их нельзя использовать в остальной программе

Глобальные переменные — переменные, записанные в самой программе. Они могут участвовать в вычислениях в любой точке программы, без риска быть стертым после их использования в локальных функциях

4. Для возврата сразу нескольких значений из функции, в операторе return можно перечислить вывод всех необходимых данным через запятую. В итоге объявляется кортеж, в который будут включены указанные данные

5. В программировании функции могут не только возвращать данные, но и принимать их. В Python данное включение переменных в функцию

выполняется посредством приписывания после названия функции в скобках названий переменных, которые потом указываются внутри функций для работы с ними. После объявления такой или таких переменных во время вызова функции глобальная переменная может быть вписана на место указанной в функции, чтобы присвоить ее значение локальной переменной. Такие переменные, указанные в скобках в названии функции, называются аргументами

6. В Python у функций могут быть параметры, чьи значения уже по умолчанию установлены. Для таких данных при вызове функции не надо передавать значения глобальных переменных, хотя возможность это сделать не исключается. Чтобы задать значение по умолчанию, в определении функции в скобках необходимо задать значение необходимому аргументу

7. Python поддерживает синтаксис, позволяющий определять небольшие однострочные функции на лету. Существуют так называемые lambda-функции, используемые везде, где требуется функция. В первую очерель, lambda — это выражение, а не инструкция, поэтому её можно вызвать даже там, где синтаксис Python не позволит использовать инструкцию def — например, в вызовах функций

8. PEP 257 описывает соглашения, связанные со строками документации Python. Цель PEP — стандартизировать структуру строк документации т.е. указать, как корректно и удобно написать код, что он должен в себя включать. Данный PEP описывает соглашения, а не правила или синтаксис

Строки документации — строковые литералы, которые являются первым оператором в модуле, функции, классе или определении метода. Такая строка документации становится специальным атрибутом __doc__ этого объекта. Все модули, как правило, должны иметь строки документации, и все функции и классы, экспортируемые модулем, также должны иметь строки документации. Публичные методы (в том числе __init__) также

должны иметь строки документации. Пакет модулей может быть документирован в __init__.py

Для согласованности, всегда нужно использовать """triple double quotes""" для строк документации. Можно использовать r"""raw triple double quotes""", если будет присутствовать обратная косая черта в строке документации. Существует две формы строк документации: однострочная и многострочная

9. Одиночные строки документации предназначены для действительно очевидных случаев. Они должны умещаться на одной строке. Однострочная строка документации не должна быть "подписью" параметров функции / метода (которые могут быть получены с помощью интроспекции). Этот тип строк документации подходит только для C функций (таких, как встроенные модули), где интроспекция не представляется возможной. Тем не менее, возвращаемое значение не может быть определено путем интроспекции.

Многострочные строки документации состоят из однострочной строки документации с последующей пустой строкой, а затем более подробным описанием. Первая строка может быть использована автоматическими средствами индексации, поэтому важно, чтобы она находилась на одной строке и была отделена от остальной документации пустой строкой. Первая строка может быть на той же строке, где и открывающие кавычки, или на следующей строке. Вся документация должна иметь такой же отступ, как кавычки на первой строке.

Вывод. В ходе выполнения работы были приобретены навыки по работе с функциями при написании программ с помощью языка программирования Python версии 3.x